Theoretical Computer Science 410 (2009) 3925–3941

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Shortest synchronizing strings for Huffman codes☆

Marek Tomasz Biskup*, Wojciech Plandowski

Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland

ARTICLE INFO

Article history:
Received 16 November 2008
Received in revised form 1 May 2009
Accepted 2 June 2009
Communicated by B. Durand

Keywords:
Huffman code
Synchronizing string
Finite automaton
Černý conjecture

ABSTRACT

Most complete binary prefix codes have a synchronizing string, that is a string that resynchronizes the decoder regardless of its previous state. This work presents an upper bound on the length of the shortest synchronizing string for such codes. Two classes of codes with a long shortest synchronizing string are presented. It is known that finding a synchronizing string for a code is equivalent to finding a synchronizing string of some finite automaton. The Černý conjecture for this class of automata is discussed.

1. Introduction

Huffman codes are the most popular variable length codes. In the presence of channel errors a large part of an encoded message can be destroyed because of the loss of synchronization between the decoder and the coder. In the case of some Huffman codes, under certain assumptions on the message source, the decoder will eventually resynchronize, and the following symbols will be decoded correctly. These codes are called *synchronizing*. Capocelli et al. [3] proved that codes are synchronizing if and only if they have a *synchronizing string* — a string such that when received by the decoder always puts it into synchronization. Freiling et al. [8] proved that almost all Huffman codes have a synchronizing string. More precisely, they proved that the probability of drawing randomly a code without a synchronizing string decreases to zero with increasing code size.

Schützenberger [16] analyzed possible distribution of codeword lengths in synchronizing prefix codes. Rudner [14] gave an algorithm for the construction of a synchronizing Huffman code for a given distribution of codeword lengths, that works under some assumptions on the distribution. His work was further extended in [12,9]. Capocelli et al. [4] showed how to modify a Huffman code by adding a little redundancy to create a synchronizing code. Ferguson and Rabinowitz [7] analyzed codes whose synchronizing string is a codeword.

The synchronization recovery of a Huffman code can be modeled with a finite automaton whose states are proper prefixes of codewords (or internal nodes of the code's tree). This automaton will be called a *Huffman automaton*. Such an automaton was used by Maxted and Robinson [11] to compute for a given code the average number of symbols lost before resynchronization.

A lot of research has been done in the area of automata synchronization. A synchronizing string for a finite automaton $\langle Q, \Sigma, \delta\rangle$ is a string s that brings all states to one particular state. That is $\delta(q_1, s) = \delta(q_2, s)$ for any states $q_1, q_2 \in Q$.

☆ This is an extended version of the paper that originally appeared in Edward Ochmanski, Jerzy Tyszkiewicz (Eds.), MFCS, in: Lecture Notes in Computer Science, vol. 5162, Springer, 2008, pp. 120–131.

* Corresponding author. Tel.: +48 600 253 491.
E-mail addresses: mbiskup@mimuw.edu.pl (M.T. Biskup), wojtekpl@mimuw.edu.pl (W. Plandowski).
URL: http://mimuw.edu.pl/~mbiskup (M.T. Biskup).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.06.005

An automaton will be called *synchronizing* if it has a synchronizing string. The famous Černý conjecture [5] states that a synchronizing finite automaton with N states has a synchronizing string of length $(N-1)^2$.

Although there are proofs for certain classes of automata, for instance in [2,10], the problem remains open. There are some bounds on the length of the shortest synchronizing string. For instance Pin [13] proved that $\frac{1}{6}(N^3-N)$ is an upper bound. Some research has also been done to find automata with long shortest synchronizing strings. Černý [5] constructed a series of automata with shortest synchronizing string of length $(N-1)^2$. Ananichev et al. [1] considered how long a synchronizing string can be if there is a letter that reduces the number of states by two. Trahtman [17] searched for worst-case automata.

Eppstein [6] gave an algorithm for testing whether an automaton is synchronizing and for the construction of a synchronizing string of length $O(N^3)$ for a synchronizing automaton. His algorithm requires $O(N^3)$ operations for a constant-size alphabet. An overview of the area of automata synchronization is given in [15].

It is rather clear that a synchronizing string for a Huffman code is also a synchronizing string for the Huffman automaton of the code, and vice versa. Nevertheless, it seems that so far both areas of research have not been related. This paper fills this gap.

First, we explain that Huffman code synchronization is equivalent to Huffman automaton synchronization. Then, we prove an upper bound on the length of the shortest *merging string* for a set of two states of a Huffman automaton: the root of the code's tree and another internal node of the tree. A merging string for a set of states is a string that brings all states of the set to the same state. The proof is constructive and an algorithm for the construction of the shortest merging string for such nodes is given. The execution of this algorithm also suffices for answering whether a code is synchronizing.

Then, we present an upper bound on the length of the shortest synchronizing string of a Huffman automaton. For most (but not all) codes the bound is better than the Černý conjecture. Also an algorithm for the construction of a synchronizing string for a Huffman automaton is presented. To the authors' best knowledge, this class of automata has not been studied yet. The bounds presented here are better than the bounds $O(N^3)$ for general automata. Both algorithms are faster than the one of Eppstein [6].

Afterwards, results of experimental search for worst-case codes are shown. Three classes of Huffman codes are presented. The codes give a lower estimate on the possible upper bounds of the length of the shortest synchronizing or merging string. It is conjectured (but, unfortunately, not proved) that these classes of codes are the worst-case codes. It is interesting that the length of their synchronizing or merging strings is much lower than the bound proved.

2. Definitions and notation

A *word* is a string of letters, for instance $w = w_0 w_1 \dots w_{k-1}$. The empty word is denoted by ϵ. The subword of a word w from the letter p to $q-1$ is denoted by $w[p..q)$. The length of a word w is denoted by $|w|$. A sequence of k letters a is denoted a^k. For instance, for $w =$ '*abc*', $w[1..2) =$ '*b*', $w[1..3) =$ '*bc*', $w[0..1) =$ '*a*', $|w| = 3$ and $0^4 =$ '0000'.

A *proper binary tree* is a tree with each node being either an *internal node* with two children, or a *leaf* node with no children. Each left outgoing edge is labeled with 0 (0-edge). Each right outgoing edge is labeled with 1 (1-edge). The root of a tree is denoted by ε.

The *label* of a node n in a proper binary tree is the string $\pi(n)$ formed of edge labels on the path from the root to n. Labels of nodes in a given tree are unique and they will be used to name the nodes. We have $\pi(\varepsilon) = \epsilon$, which explains the notation. The label of the left child of the root is 0 and the label of the right child of the root is 1.

The number of leaves in a tree is denoted by N. The height of a tree is denoted by h. In this paper, a code C such that $C = \{\pi(n) | n \text{ is a leaf of } T\}$ for some proper binary tree T, is called a *Huffman code*. The tree T is called a *Huffman tree*.

Let a *Huffman Automaton* $\mathcal{T}$ be an automaton whose states are internal nodes of the Huffman tree T. The transition function $\delta(n, b)$, $b \in \{0, 1\}$, brings an automaton from the node n to its b-edge child, if it is not a leaf, or to the root otherwise. The function δ can be extended by induction to strings: $\delta^*(q, b_0 \dots b_{k-1}) = \delta(\delta^*(q, b_0 \dots b_{k-2}), b_{k-1})$ and $\delta^*(q, \epsilon) = q$. This extension δ^* will also be denoted by δ. For a subset $\mathscr{C}$ of states of a Huffman automaton we denote, $\delta(\mathscr{C}, a) := \{\delta(q, a) | q \in \mathscr{C}\}$.

We say that a word w *brings* a node n to a node n' if $n' = \delta(n, w)$. Then n' is the result of *applying* w to n. In addition, we say that w brings a node n to a leaf if $\delta(n, w) = \varepsilon$ and w is not empty. This is justified because the construction of the Huffman automaton $\mathcal{T}$ may be seen as merging the leaves of the tree T with the root of T. We say that w brings a node n to n' *without loops* if none of the nodes $\delta(n, w[0, 1)), \delta(n, w[0, 2)), \dots, \delta(n, w[0, |w|-2))$ is the root.

The values $T, \mathcal{T}, \delta, N, h, \varepsilon, \pi$ depend on the code C. We assume that it is always clear from the context which code (or, equivalently, which Huffman tree) is being considered.

Definition 1 (*[16]*). A *synchronizing string* for a Huffman code is a string s such that ws is a sequence of codewords for any binary word w.

Equivalently, a synchronizing string is a string that brings any node of the Huffman automaton to the root.

Definition 2. Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be a finite automaton. A *synchronizing string* for $\mathcal{A}$ is a word w such that $|\delta(Q, w)| = 1$. A *merging string* for a set of states $\mathscr{R} \subseteq Q$ of the automaton $\mathcal{A}$ is a word w such that $|\delta(\mathscr{R}, w)| = 1$.

Definition 3. Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be a finite automaton. The *power automaton* for $\mathcal{A}$ is the automaton $\mathcal{P}(\mathcal{A}) = (\mathcal{P}(Q), \Sigma, \delta_{\mathcal{P}})$, where $\mathcal{P}(Q)$ denotes the set of all subsets of Q and $\delta_{\mathcal{P}}(\mathscr{R}, a) = \delta(\mathscr{R}, a)$ for $\mathscr{R} \in \mathcal{P}(Q)$.

The states of the power automaton $\mathcal{P}(\mathcal{A})$ are sets of states of the automaton $\mathcal{A}$. They are called *configurations* and denoted by capital script letters, e.g. $\mathscr{A}, \mathscr{B}, \mathscr{C}, \ldots$.

The operation of the power automaton $\mathcal{P}(\mathcal{A})$ can be seen as movements of coins that lie on some states of the automaton $\mathcal{A}$. If the power automaton is in configuration $\mathscr{C}$, the coins lie on the states $q \in \mathscr{C}$. Then, if the power automaton makes a transition by a letter a, the coins move accordingly to the transition function δ for the automaton $\mathcal{A}$ — a coin on a state p moves onto the state $\delta(p, a)$. If more than one coin goes to the same state, only one of them is kept.

It is easy to see that after applying the letter a to configuration $\mathscr{C}$, the set of states with coins is exactly $\delta_{\mathcal{P}}(\mathscr{C}, a)$. This analogy helps to visualize the operation of a power automaton and gives some intuition. For instance, the string w is synchronizing if and only if applying w to the automaton $\mathcal{A}$ with a coin on each state results in just one coin remaining.

An automaton is *synchronizing* if it has a synchronizing string. A Huffman code is *synchronizing* if it has a synchronizing string.

Theorem 4. *A synchronizing string for a Huffman code C is a synchronizing string for the Huffman automaton $\mathcal{T}$ of the code. A synchronizing string s for the Huffman automaton $\mathcal{T}$, such that s brings all nodes to the root, is a synchronizing string for the Huffman code.*

The proof of the theorem is easy and may be omitted. The theorem states that a Huffman code is synchronizing if and only if its Huffman automaton is synchronizing.

3. Merging string for a pair of states

Theorem 5. *Let C be a synchronizing Huffman code of size N, let T and $\mathcal{T}$ be, respectively, the Huffman tree and the Huffman automaton for C. For any node n of $\mathcal{T}$ there is a merging string s_n for the set $\{n, \varepsilon\}$ (ε is the root of T), with*

$$|s_n| \leq \sum_{p \in I(T) \setminus \{\varepsilon\}} h_p, \tag{1}$$

where $I(T)$ is the set of the internal nodes of T and h_p is the height of the subtree of T rooted at p.

Proof. C is synchronizing so $\mathcal{T}$ has a synchronizing string $s_{\mathcal{T}}$. Let us consider a merging string s_n for n and ε of minimal length. It exists, because $s_{\mathcal{T}}$ merges ε and n, but it need not be unique. The string s_n brings both nodes to the root, because otherwise we could remove the last letter of s_n and the result would still merge n and ε.

Let $\{n_i, m_i\}$ be the unordered pairs of nodes that appear when consecutive prefixes of s_n are applied to the initial configuration $\{n, \varepsilon\}$:

$$\{n_i, m_i\} = \delta\left(\{n, \varepsilon\}, s_n[0..i)\right), \quad i = 0, \ldots, |s_n|. \tag{2}$$

In particular, $\{n_0, m_0\} = \{n, \varepsilon\}$ and $\{n_{|s_n|}, m_{|s_n|}\} = \{\varepsilon\}$ (a singleton is also considered as a pair).

Consider the subsequence $\{n_{i_k}, \varepsilon\}_{k=0,\ldots,\ell}$ of $\{n_i, m_i\}_{i=0,\ldots,|s_n|}$ which is formed of pairs containing the root. Each node p, appears in this subsequence as the partner of ε at most once, because pairs do not repeat in $\{n_i, m_i\}$ (otherwise we could shorten the string s_n). The string $s_n[i_k, i_{k+1})$, that brings $\{n_{i_k}, \varepsilon\}$ to $\{n_{i_{k+1}}, \varepsilon\}$, is a string that either brings the node n_{i_k} to a leaf without loops or that brings ε to a leaf without loops. In either case the length of $s_n[i_k, i_{k+1})$ is at most $h_{n_{i_k}}$ (note that in the second case the node $n_{i_{k+1}}$ is in the subtree of n_{i_k}, unless $n_{i_{k+1}} = \varepsilon$). It follows that

$$|s_n| = \sum_{k=0}^{\ell-1} |s_n[i_k, i_{k+1})| \leq \sum_{k=0}^{\ell-1} h_{n_{i_k}} \leq \sum_{p \in I(T) \setminus \{\varepsilon\}} h_p. \tag{3}$$

The last inequality follows from the fact that n_{i_k} are different nodes of T. The value of h_ε is not counted in the sum because the set $\{\varepsilon\}$ appears only as the last element of the sequence $\{n_{i_k}, \varepsilon\}$. □

Let H_T be the value of the bound in Theorem 5. H_T is the sum of heights of all the nontrivial subtrees of T apart from the whole tree. We will compare H_T with Π_T — the sum of depths of all the internal nodes, with W_T — the sum of depths of all the leaves of T (that is the sum of codeword lengths), and with S_T — the sum of the number of leaves in all subtrees of T. The proof of Theorem 5 can be easily modified to prove that $|s_n|$ does not exceed Π_T, W_T and S_T. It turns out that the bound $|s_n| \leq H_T$ is the best of the four.

Lemma 6. *Let T be a proper binary tree, let $I(T)$ be the set of the internal nodes of T, let $L(T)$ be the set of leaves of T, let h_n and N_n be, respectively, the height and the number of leaves in the subtree rooted at the node n of T, let $|\pi(n)|$ be the distance from the root to n and let N_T be the number of leaves in T. Let us define:*

$$H_T = \sum_{n \in I(T) \setminus \{\varepsilon\}} h_n, \qquad \Pi_T = \sum_{n \in I(T)} |\pi(n)|, \tag{4}$$

$$W_T = \sum_{n \in L(T)} |\pi(n)|, \qquad S_T = \sum_{n \in I(T) \cup L(T)} N_n. \tag{5}$$

(a) Huffman tree 1. (b) Huffman tree 2. (c) Huffman tree 3.

Fig. 1. Illustration for Example 9.

Then the following holds:

$$\Pi_T = W_T - 2N_T + 2, \tag{6}$$
$$W_T = S_T - N_T, \tag{7}$$
$$H_T \le \Pi_T \le W_T \le S_T. \tag{8}$$

Proof. Note that the index of N can be either a node or a tree. This should not cause confusion as there is a 1–1 correspondence between nodes and trees — a node n corresponds to the tree T rooted at n with $N_T = N_n$.

We prove the equations and the inequalities by induction. For a tree that consist of just the root node the values H_T, Π_T and W_T are equal to 0, S_T and N_T are equal to 1 and (6), (7), (8) are correct. If the tree T consists of two subtrees T_1 and T_2 joined in a common root, the following recurrences hold:

$$H_T = H_{T_1} + H_{T_2} + h_{T_1} + h_{T_2}, \tag{9}$$
$$\Pi_T = \Pi_{T_1} + \Pi_{T_2} + (N_{T_1} - 1) + (N_{T_2} - 1), \tag{10}$$
$$W_T = W_{T_1} + W_{T_2} + N_{T_1} + N_{T_2}, \tag{11}$$
$$N_T = N_{T_1} + N_{T_2}, \tag{12}$$
$$S_T = S_{T_1} + S_{T_2} + N_{T_1} + N_{T_2}, \tag{13}$$

where h_{T_i} is the height of the tree T_i.

Eq. (6) can be proved by induction, by substituting the induction hypothesis (6) for Π_{T_1} and Π_{T_2} to (10) and using (11) and (12):

$$\begin{aligned}\Pi_T &= \Pi_{T_1} + \Pi_{T_2} + N_{T_1} + N_{T_2} - 2\\ &= (W_{T_1} - 2N_{T_1} + 2) + (W_{T_2} - 2N_{T_2} + 2) + N_{T_1} + N_{T_2} - 2\\ &= (W_{T_1} + W_{T_2} + N_{T_1} + N_{T_2}) - 2(N_{T_1} + N_{T_2}) + 2\\ &= W_T - 2N_T + 2.\end{aligned}$$

The recurrences for W_T and $S_T - N_T$ are identical:

$$S_T - N_T = (S_{T_1} - N_{T_1}) + (S_{T_2} - N_{T_2}) + N_{T_1} + N_{T_2} \tag{14}$$

and the value of $(S_T - N_T)$ for the tree with one node is 0, so $S_T - N_T = W_T$ holds for all trees. The inequality $h_T \le N_T - 1$ together with the recurrences (9) and (10) prove the first inequality of (8). Other inequalities follow from (6) and (7). □

Corollary 7. *Let w_i be all the codewords of a Huffman code. Then*

$$|s_n| \le \sum_i |w_i| \quad \text{and} \quad |s_n| \le (N-2)(h-1). \tag{15}$$

The result of Theorem 5 can be improved if we notice that the sequence $\{n_{i_k}, \varepsilon\}$, defined in the proof of Theorem 5, cannot contain two nodes n_{i_k} and $n_{i_{k'}}$ that are roots of identical subtrees of T. Indeed, otherwise we could shorten the string s_n by cutting off the substring that brings n_{i_k} to $n_{i_{k'}}$. This gives the following result:

Corollary 8. *The bound of Theorem 5 can be improved to:*

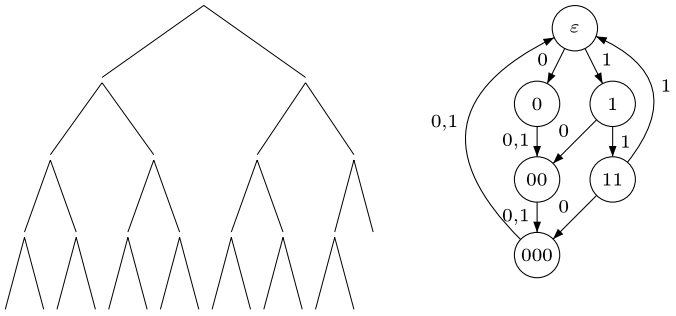
$$|s_n| \le \sum_{t \in \mathfrak{T}(T) \setminus \{T\}} h_t, \tag{16}$$

where $\mathfrak{T}(T)$ is the set all distinct subtrees of T and h_t is the height of the tree t.

Example 9. Let us consider the tree from Fig. 1(a). It has two distinct subtrees other than the whole tree: one of height 1 and one of height 2. It follows that for each node n there is a merging string with ε of length at most 3. For instance, for the node 0 the merging string is 110.

The tree from Fig. 1(b), has exactly the same subtrees, so the bound on the length of its merging strings is the same. In this case the node 01 requires a merging string of length at least 3, for instance 011.

Note that the bound is not sharp and for some trees all nodes have shorter merging strings. For instance, each node of the tree from Fig. 1(c) can be merged with the root using a string of length 1 (the string '1'). The bound from Corollary 8 for this tree is 3.



(a) Huffman tree. (b) Minimized Huffman automaton.

Fig. 2. A Huffman tree and its minimized Huffman automaton.

The idea of identifying common subtrees can be formalized by introducing the *minimized Huffman automaton*. Although this does not give here a better estimate of the length of the shortest merging string for the set $\{n, \varepsilon\}$, it is interesting in itself. This construction will also be used later in this paper.

Definition 10. A *minimized Huffman automaton* for a Huffman code C is an automaton made of the Huffman automaton for C by merging the states that are roots of identical subtrees of the Huffman tree T for C.

It is easy to see that minimized Huffman automata have exactly two edges, labeled with 0 and 1, going out of each node, thus their transition function is indeed a function. An example of a minimized Huffman automaton is presented in Fig. 2.

We will say that a set $\mathscr{R}$ of states of a Huffman automaton $\mathcal{T}$ corresponds to the set $\mathscr{R}_m$ of states of the minimized Huffman automaton $\mathcal{T}_m$ if $\mathscr{R}_m$ is the smallest set satisfying: if $q \in \mathscr{R}$ and q is merged to a state q' of $\mathcal{T}_m$ then $q' \in \mathscr{R}_m$.

Theorem 11. *Let C be a synchronizing Huffman code, let $\mathcal{T}$ be the Huffman automaton for C and let $\mathcal{T}_m$ be the minimized Huffman automaton for C. Let $\mathscr{R}$ be a set of states of $\mathcal{T}$ and let $\mathscr{R}_m$ be the corresponding set of states of $\mathcal{T}_m$. If s is a merging string for $\mathscr{R}$ then s is a merging string for $\mathscr{R}_m$. If s' is a merging string for $\mathscr{R}_m$ that brings all nodes of $\mathscr{R}_m$ to the root then s' is a merging string for $\mathscr{R}$.*

Proof. Let us consider coins on the states of the automata $\mathcal{T}$ and $\mathcal{T}_m$. Moving coins according to a string w and then merging the states of $\mathcal{T}$ to get its minimized version (with removing duplicate coins on the same state) is equivalent to merging the states first and then moving coins. This observation proves the theorem. Indeed, applying s to $\mathscr{R}$ leaves only one coin. Then merging the states does not multiply the coins. Applying s' to $\mathscr{R}_m$ leaves only the coin in the root. Thus, applying s' to $\mathscr{R}$ may leave a coin only in the root of $\mathcal{A}$, because no other state was merged with the root of $\mathcal{T}$ to be the root of $\mathcal{T}_m$. □

Note that the minimized Huffman automaton is implicitly used in Corollary 8, where we consider all non-identical subtrees of a Huffman tree. The roots of such subtrees are states of the minimized Huffman automaton.

Theorem 5 leads to an algorithm for finding the shortest merging string for a set $\{n_0, \varepsilon\}$, where n_0 is any state of $\mathcal{T}$. First a graph $G = (V, E)$ is created. The vertices of G are unordered pairs $\{n, \varepsilon\}$, where n is a state of $\mathcal{T}$. The edges of G are weighted; $\{n_1, \varepsilon\} \to \{n_2, \varepsilon\}$ is an edge if there is a string w that brings $\{n_1, \varepsilon\}$ to $\{n_2, \varepsilon\}$ without passing through any other pair $\{n, \varepsilon\}$. The weight of the edge is the length of the shortest such string w (note that the string w need not be unique).

Such a string w is also the label of the edge $\{n_1, \varepsilon\} \to \{n_2, \varepsilon\}$, although it will not be stored explicitly. Instead, for retrieving the label w, we will store a mark M. The mark will depend on the target pair of the edge. If the target is a pair $\{n_2, \varepsilon\}$ with $n_2 \neq \varepsilon$, the mark is equal to either n_1 if $n_2 = \delta(n_1, w)$, or to ε if $n_2 = \delta(\varepsilon, w)$. In either case $n_2 = \delta(M, w)$, the node n_2 is in the subtree of the node M and w is formed of labels on the path from M to n_2. If the target of an edge is a singleton $\{\varepsilon\}$, that is $n_2 = \varepsilon$, the mark M is the leaf $\delta(\varepsilon, w)$. In this case the word w is formed of labels on the path from ε to M.

The construction of the graph requires DFS-traversing the Huffman tree with a pair of nodes $\{n_1, n_2\}$, starting at $\{n, \varepsilon\}$ and applying transitions of the Huffman automaton to both nodes of the pair. The traversal goes forward until a set $\{n', m\}$ is reached, with m being a leaf. Then, the edge $\{n, \varepsilon\} \to \{n', \varepsilon\}$ is added to the graph with the number of steps from $\{n, \varepsilon\}$ to $\{n', m\}$ as its weight. If such an edge has been added before, only the weight is updated to be the minimum of the previous weight and the new one. Finally, the mark M of the edge is set appropriately.

The cost of processing each pair $\{n, \varepsilon\}$ during the construction of the graph G is proportional to the size of the subtree rooted at n, because the DFS traversal is limited to the subtree of n. The size of a tree is proportional to the number of its leaves. It follows that the construction of G uses the time proportional to the sum of the number of leaves in all subtrees of T, S_T. By Lemma 6, this is $O(\sum |w_i|)$, where w_i are the codewords given by the tree T. The number of vertices in the graph is $|V| = N - 1$. The number of edges is also bounded by the sum of sizes of all the subtrees of the tree, that is $|E| = O(\sum |w_i|)$.

The shortest merging string for a set $\{n, \varepsilon\}$ is given by the lightest path from $\{n, \varepsilon\}$ to $\{\varepsilon\}$ in the graph G. The tree of the lightest paths from any node to $\{\varepsilon\}$ can be constructed using Dijkstra's algorithm in $O(|E| + |V| \log |V|)$. Since $|V| = O(N)$, $|E| = O(\sum |w_i|)$ and $\sum |w_i| \geq N \log N$, the lightest paths' tree can be computed in $O(\sum |w_i|)$.

To print out the shortest merging string for a set $\{n, \varepsilon\}$ it is necessary to reconstruct the labels of each edge. For an edge $\{n, \varepsilon\} \to \{n', \varepsilon\}$, where $n' \neq \varepsilon$, we may traverse the tree up from n' until we reach the node M (the mark of the edge). The

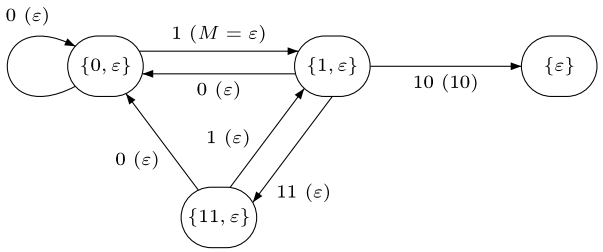


Fig. 3. The graph from the algorithm for finding the shortest merging strings for the code from Fig. 1(a); the mark M is given in parenthesis.

string w is formed of labels on the path that goes from M down to n'. The word can be computed in the time proportional to its length. For an edge $\{n, \varepsilon\} \to \{\varepsilon\}$ we do the same, but we traverse the tree from M up to ε.

The properties of the algorithm can be summarized as follows.

Theorem 12. *Let $\mathcal{T}$ be a Huffman automaton. The algorithm for computing the shortest merging string for a set $\{n, \varepsilon\}$, where n is any state of $\mathcal{T}$, requires $O(\sum_i |w_i|)$ preprocessing time. Then, the shortest merging string for each $\{n, \varepsilon\}$ pair can be found in time proportional to the length of the merging string.*

Example 13. Fig. 3 presents the graph G for the code from Fig. 1(a). There are four internal nodes in the tree, so there are four nodes in the graph. Each edge in the graph is marked with the word w that brings a state to the other one, and also with the mark M, in parenthesis. The shortest merging strings can easily be read from the graph. For the set $\{0, \varepsilon\}$ it is 110, for $\{1, \varepsilon\}$ — 10, and for $\{11, \varepsilon\}$ — 110.

4. Length of a synchronizing string

In this section we give an upper bound on the length of the shortest synchronizing string for any synchronizing Huffman code. We begin with a lemma that helps to prove the main theorem of this section (Theorem 16).

Lemma 14. *Let T be a proper binary tree with N leaves. There exists a string w of length at most $\lceil \log N \rceil$ such that for each node n of T some prefix of w labels a path from n to a leaf.*

Proof. Let us assume the contrary: for any string w of length $|w| = \lceil \log N \rceil$, there is a node n_w such that no prefix of w brings it to a leaf. Let $m_w = \delta(n_w, w)$. Then m_w is an internal node of the tree and w is a suffix of $\pi(m_w)$.

For two strings, $w_1 \neq w_2$, of length $\lceil \log N \rceil$ the nodes m_{w_1} and m_{w_2} are different. Indeed, the suffixes of $\pi(m_{w_1})$ and $\pi(m_{w_2})$ of length $\lceil \log N \rceil$ are different as they are equal to w_1 and w_2, respectively. But there are $N - 1$ internal nodes of T and $2^{\lceil \log N \rceil} \geq N$ strings w. The contradiction proves that the initial assumption is wrong. □

Example 15. Lemma 14 states that there is a string of length at most $\lceil \log 5 \rceil = 3$ that moves each node of the tree for the code from Fig. 1(a) through a leaf. Indeed, the string 00 is such a string.

Theorem 16. *For any synchronizing Huffman code of size N the length of the shortest synchronizing string s is at most*

$$|s| \leq \lceil \log N \rceil + (\lceil \log N \rceil - 1)X = O(Nh \log N), \tag{17}$$

where h is the length of the longest codeword, and

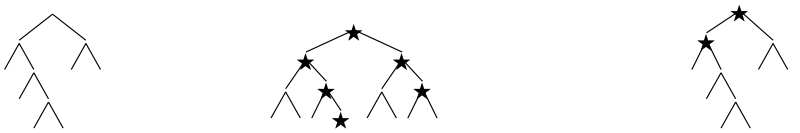
$$X = \sum_{t \in \mathfrak{T}(T) \setminus \{T\}} h_t, \tag{18}$$

and $\mathfrak{T}(T)$ is the set all different subtrees of T, and h_t is the height of the tree t.

Proof (*Sketch*). We first find a string w given by Lemma 14 and apply it to coins on all states of $\mathcal{T}$. It reduces the number of coins to at most $\lceil \log N \rceil$, because the final position of each coin is determined by some proper suffix of w. We may assume that one of the coins is on the root of $\mathcal{T}$ (otherwise w could be shortened). Then, we pick a node n with a coin and we construct the shortest string s_n that merges n and the root (to get shorter synchronizing strings it is better to pick a node n with the shortest merging string for $\{n, \varepsilon\}$ among the nodes with a coin). By Corollary 8, $|s_n| \leq X$. This string applied to the current set of coins reduces the number of coins by at least one. Repeating this procedure additional $(\lceil \log N \rceil - 2)$ times leaves us with just one coin. The length of the merging string does not exceed $\lceil \log N \rceil + (\lceil \log N \rceil - 1)X$. Finally, $X < Nh$ gives the asymptotic bound $O(Nh \log N)$. □

The proof of Theorem 16 is constructive and gives an algorithm for the construction of a synchronizing string for a Huffman code. The algorithm works as follows.

First, a string w from Lemma 14 is found. It is done by checking all the $O(N)$ strings of length less than or equal to $\lceil \log N \rceil$ in the following way. For each node n of T, its subtree is DFS-traversed. Each time an internal node m that is not farther than $\lceil \log N \rceil$ steps below n is reached, the string w on the path from n to m is marked as BAD. This means that no prefix of



(a) A tree.

(b) Strings of length ≤ 3 presented on a full binary tree; the strings that do not bring all nodes of the tree from Fig. 4(a) through a leaf are marked with a star.

(c) The result of applying the string 00 to the set of all states of the tree.

Fig. 4. Illustration for the algorithm of finding a synchronizing string for a code.

w brings n to a leaf. The set of all strings of length less than or equal to $\lceil \log N \rceil$ can be stored in a full binary tree of height $\lceil \log N \rceil$.

After the traversal, the strings that have not been marked as BAD bring any node through a leaf. By Lemma 14, there is at least one such a string of length $\lceil \log N \rceil$ or less. In fact, with this approach the shortest such string is found. The cost of this phase is proportional to the sum of sizes of all subtrees of T, which is $O(S_T) = O(\sum_i |w_i|)$, by Lemma 6.

After finding the string w we can apply it to the set of all internal nodes of T. This takes $O(N \log N)$ time. Then, at most $\log N$ merging strings for $\{n, \varepsilon\}$, with some node n, suffice to build a synchronizing string. Computing the merging strings requires $O(\sum_i |w_i|)$ preprocessing time and then any string can be read in the time proportional to its length (Theorem 12). The length of each merging string is bounded by X and there are at most $\log N$ vertices that have to be moved using each such string. Thus the total cost of the algorithm is $O(X \log^2 N + \sum_i |w_i|)$.

Theorem 17. *Let C be a Huffman code of size N. The time complexity for the algorithm that computes a synchronizing string for C is*

$$O(X \log^2 N + \sum_i |w_i|), \tag{19}$$

where X is defined as in Theorem 16 and w_i are codewords of C.

Example 18. Let us consider the tree from Fig. 4(a). There should be a string of length at most 3 that brings all nodes through a leaf. Such a string can be found with the algorithm just described. Fig. 4(b) shows the tree of strings of length at most 3. The strings that were marked as BAD are starred. The shortest nodes that are not starred are 00 and 10. Both strings bring all nodes through a leaf. Let us chose 00.

Fig. 4(c) shows the coins that remain after applying the string 00 to the tree with coins on all states. These two nodes with coins can be merged with a string of length at most 6, according to Corollary 8. In fact, the string 10 is the shortest merging string for these two nodes. The synchronizing string for the tree found by the algorithm is 0010. This is not the shortest synchronizing string, for instance 010 is a shorter synchronizing string for this tree.

From the proof of Theorem 16 it follows that if for each pair $\{n, \varepsilon\}$ there is a merging string then the code has a synchronizing string. This also gives an algorithm to test whether a synchronizing string for a code exists, because the existence of the merging strings can be checked with the algorithm of Section 3.

Theorem 19. *The complexity for the algorithm for testing if a code has a synchronizing string is $O(\sum_i |w_i|)$.*

5. Worst-case trees

Numerical search was performed to find:

- the worst-case trees in terms of the length of the shortest synchronizing string,
- the worst-case trees in terms of the length of the shortest merging strings for a pair $\{n, \varepsilon\}$, where n is an internal node of the tree.

All trees with the number of leaves, N, from 3 to 20 were analyzed first. Then the procedure was repeated for all trees of heights, h, from 2 to 5.

5.1. Long synchronizing string

The worst-case trees for a fixed number of nodes, N, for $N = 3 \ldots 12$, are shown in Fig. 5.

In most of the tested cases the worst-case trees for fixed N, were unique up to the reflection across the y axis (relabeling 0-edges to 1-edges and 1-edges to 0-edges). The exceptions were the trees with 7 nodes — three nonequivalent trees, 10 nodes — 5 trees, and 12 nodes — 2 trees. For trees with 9, 11 and from 13 to 20 nodes the unique worst-case tree corresponds to one of the codes C_k, defined below. The codes C_k also form one of the worst-case trees with 7, 10 and 12 nodes. They can be described by the following set of codewords:

$$C_k = \{00, 010, 011, 110, 111\} \cup \{10^i 1 | i = 1, 2, \ldots, k-1\} \cup \{10^k\}, \quad k \geq 1. \tag{20}$$

The size of the code C_k is $k + 5$. The structure of these trees is shown in Fig. 6(a) and examples can be found in Fig. 5(g), (i), (m), (o), (p).

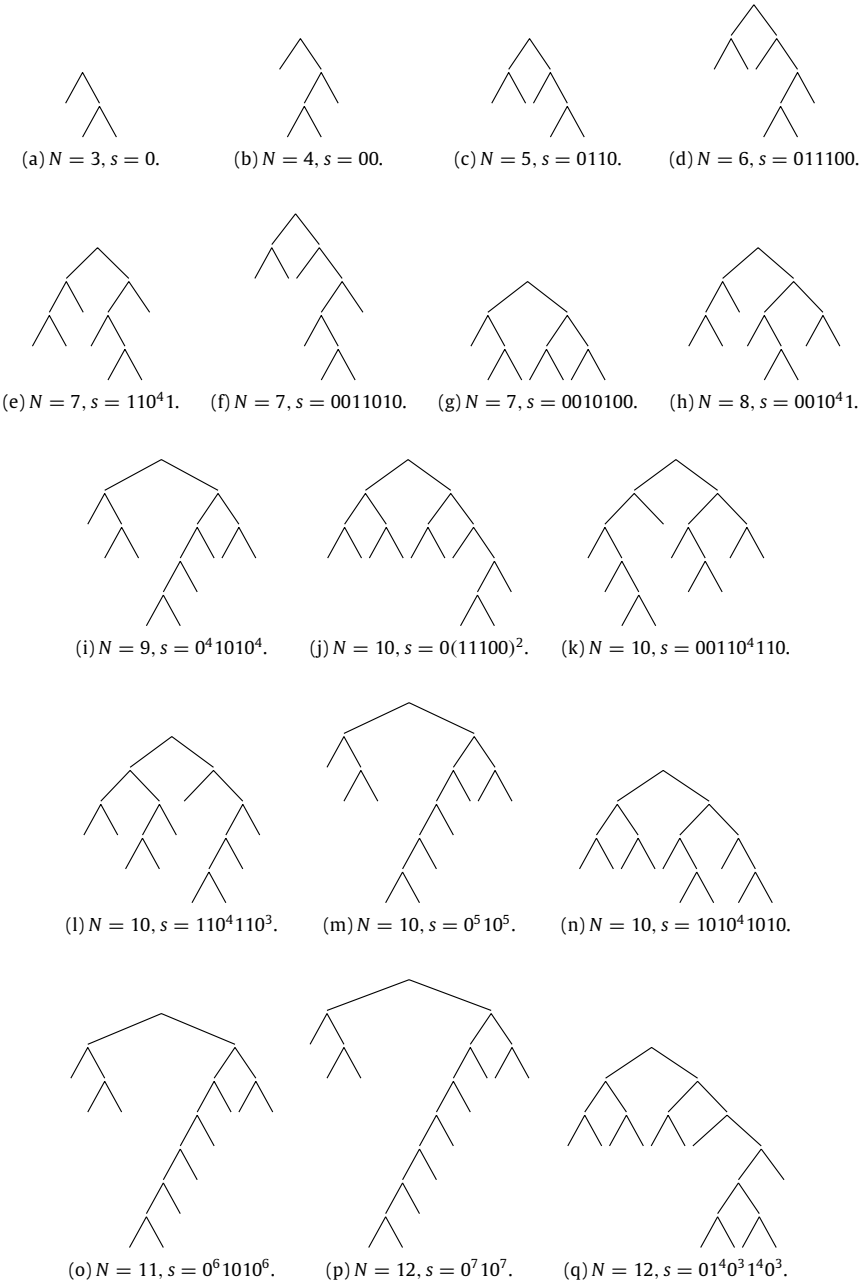


Fig. 5. Trees with longest synchronizing string for a given number of nodes, N. The synchronizing string is denoted by s.

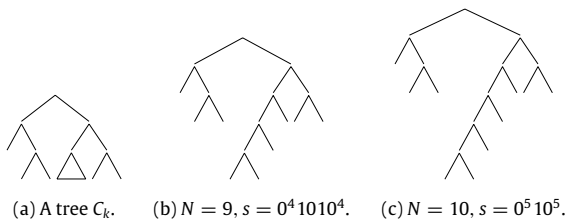


Fig. 6. The class C_k, of the worst-case trees in terms of the length of the shortest synchronizing string for a given number of nodes, N. The synchronizing string is denoted by s. The triangle denotes a code $\{1, 01, 001, \ldots, 0^i 1, 0^i 0\}$, $i = 0, 1, \ldots$.

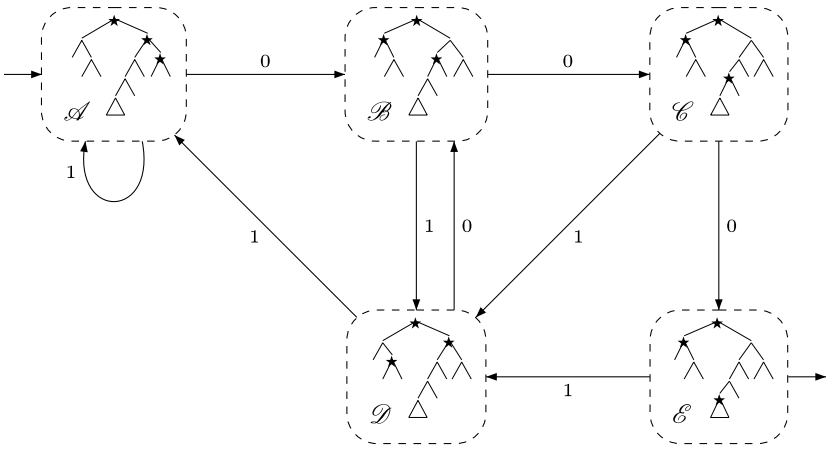


Fig. 7. A fragment of the power automaton for the trees C_k. The coins on nodes of the automaton are marked with a star. Only selected three-state configurations are shown.

Theorem 20. *The shortest synchronizing string for the tree C_k, $k \geq 1$, is $s_0 = 0^k10^k$ for odd k (even number of codewords) and $s_1 = 0^k1010^k$ or $s_2 = 0^k1110^k$ for even k (odd number of codewords). The length of the shortest synchronizing string is $2N - 9$ for even code size, N, and $2N - 7$ for odd code size.*

Proof. We will first find a merging string for the set of states $\mathscr{A} = \{\epsilon, 1, 11\}$. We will look at the power automaton by placing coins on states of the automaton $\mathcal{T}$ for the code. Assume for a moment that $k > 3$. The transitions of coins under the letters 0 and 1 for selected three-coin configurations are presented in Fig. 7. Configuration $\mathscr{A}$ is the initial configuration. All strings that lead to a configuration with less than three coins have to pass through the configurations $\mathscr{B}$, $\mathscr{C}$ and $\mathscr{E}$. The first configuration with two coins that may appear after starting from $\mathscr{A}$ is $\mathscr{L} = \{\epsilon, 0\}$. The shortest string leading from $\mathscr{A}$ to $\mathscr{L}$ is 0^k and it is unique.

Fig. 8 shows a fragment of the power automaton with two-coin configurations. The strings that reduce the number of coins to one have to pass through either $\mathscr{N}$, for odd k, or through $\mathscr{P}$, for even k.

The shortest string leading from $\mathscr{L}$ to $\mathscr{N}$ is 1000 and it is unique. Then, for odd k, the word 0^{k-3} merges the two coins of the configuration $\mathscr{N}$. The shortest merging string for the configuration $\mathscr{A}$ is then $s_0 = 0^k10^k$.

There are two shortest strings from $\mathscr{L}$ to $\mathscr{P}$: 11100 and 10100. Then, for even k, the two coins of $\mathscr{P}$ can be merged with 0^{k-2}. The shortest merging strings for the configuration $\mathscr{A}$ are, in this case, $s_1 = 0^k1010^k$ and $s_2 = 0^k1110^k$.

The proof is correct so far for $k > 3$, but it is easy to verify that s_0 is the shortest merging string for the configuration $\mathscr{A}$ for $k = 1$ and $k = 3$, and that s_1 and s_2 are the shortest merging strings for $\mathscr{A}$ if $k = 2$.

To finish the proof it is necessary to show that s_0 is a synchronizing string for C_k if k is even, and that s_1 and s_2 are synchronizing strings for C_k if k is odd. It is easy to see that applying 0^k, which is a prefix of each of the strings s_0, s_1 and s_2, to the configuration with coins on all states always results in the configuration $\mathscr{L}$, no matter if k is even of odd. This is the same as applying 0^k to $\mathscr{A}$, so these strings synchronize C_k.

The length of the synchronizing string follows easily from its form and from $k = N - 5$. □

The worst-case trees for fixed height, h, with $h = 2, 3, 4$ and 5, are shown in Fig. 9(a), (b), (c) and (d). These are full binary trees with two edges in the lower-right corner removed. They can be described by the following set of codewords:

$$D_h = \left(\{0, 1\}^h \setminus \{1^{h-1}1, 1^{h-1}0\}\right) \cup \{1^{h-1}\}, \quad h \geq 2. \tag{21}$$

The general scheme for the trees D_h is depicted in Fig. 9(e). The number of codewords in the code D_h is $2^h - 1$. For $h = 2, 3, 4, 5$, the trees D_h are unique worst-case trees up to the reflection across the y axis.

Theorem 21. *The shortest synchronizing string for the tree D_h, $h \geq 2$, is*

$$s = (1^{h-1}0^h)^{h-2}1^{h-1}, \tag{22}$$

with $|s| = 2h^2 - 4h + 1$ (however, the shortest synchronizing string is not unique).

The proof of Theorem 21 is long and will be split into several steps.

First, we will find the shortest merging string $s_{\mathscr{L}_h}$ for the set of nodes on the leftmost path of the tree, i.e. for the set $\mathscr{L}_h = \{\epsilon, 0, 00, \ldots, 0^{h-1}\}$. No synchronizing string for the tree D_h can be shorter than $s_{\mathscr{L}_h}$. We will later show that $s_{\mathscr{L}_h}$ is also a synchronizing string for D_h.

Let $\mathcal{F}_h$ be the minimized Huffman automaton for D_h, constructed from the automaton for D_h by merging states with the same subtrees. Let $\mathscr{L}'_h$ be the set of states of $\mathcal{F}_h$ that corresponds to $\mathscr{L}_h$. From Lemma 21 we know that if s is a merging string for the set $\mathscr{L}'_h$ in the automaton $\mathcal{F}_h$ and if s moves the coins to the root, it is also a merging string for the set $\mathscr{L}_h$ in the automaton D_h.

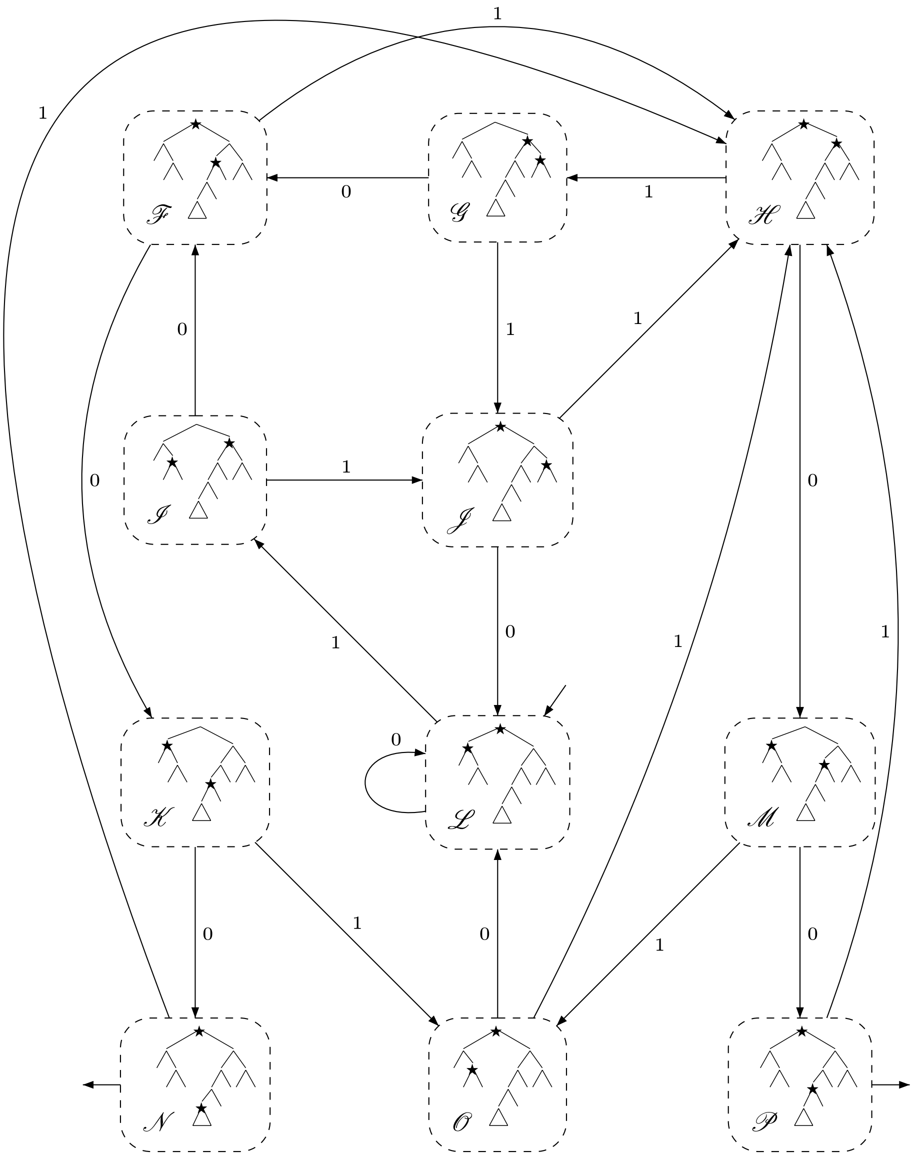
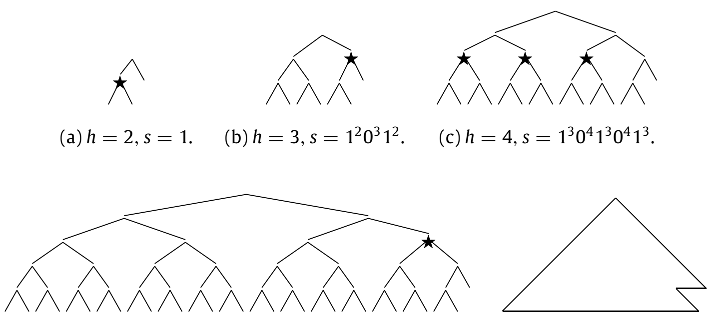


Fig. 8. A fragment of the power automaton for the trees C_k. Coins are marked with a star. Only selected two-coin configurations are shown.



(a) $h = 2$, $s = 1$. (b) $h = 3$, $s = 1^2 0^3 1^2$. (c) $h = 4$, $s = 1^3 0^4 1^3 0^4 1^3$.

(d) $h = 5$, $s = 1^4 0^5 1^4 0^5 1^4 0^5 1^4$. (e) The scheme for D_h.

Fig. 9. Trees with the worst-case length of a synchronizing and merging string among trees of fixed height, h, for $h = 2, 3, 4, 5$, and a scheme of these trees. The nodes n with the longest merging string for $\{n, \varepsilon\}$ are marked with a star.

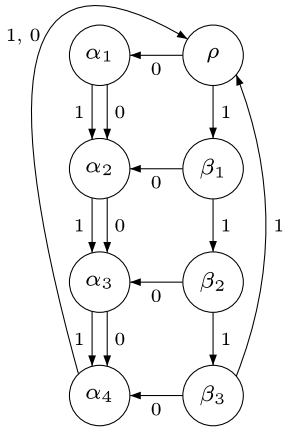


Fig. 10. The automaton $\mathcal{F}_5$ equivalent to the tree D_5.

The automaton $\mathcal{F}_5$ for the tree D_5 is depicted in Fig. 10 (see also Fig. 9(d)).

It is easy to see that the general automaton $\mathcal{F}_h$, $h \geq 3$, has nodes $\rho, \alpha_1, \ldots, \alpha_{h-1}$ and $\beta_1, \ldots, \beta_{h-2}$ (see Fig. 10) and its transitions are similar to the ones of $\mathcal{F}_5$. The nodes of $\mathcal{F}_h$ correspond to the following nodes of the Huffman tree.

$$\rho \to \{\epsilon\}$$

$$\alpha_1 \to \{0\}$$
$$\alpha_2 \to \{00, 01, 10\}$$
$$\ldots$$
$$\alpha_i \to \{0+1\}^i \setminus \{1^i\}, \quad (i = 1 \ldots, h-1)$$
$$\ldots$$

$$\beta_1 \to \{1\}$$
$$\beta_2 \to \{11\}$$
$$\ldots$$
$$\beta_j \to \{1^j\}, \quad (j = 1, \ldots, h-2),$$
$$\ldots$$

The set $\mathcal{L}'_h$ is equal to $\{\rho, \alpha_1, \alpha_2, \ldots, \alpha_{h-1}\}$.

Let us fix h. Let the sets (configurations) $\mathcal{B}_i$ of states of $\mathcal{F}_h$ be defined by:

$$\mathcal{B}_i = \delta_{\mathcal{F}_h}(\mathcal{L}'_h, 1^i), \quad i = 0, 1, \ldots, h-1, \tag{23}$$

where $\delta_{\mathcal{F}_h}$ is the transition function for the automaton $\mathcal{F}_h$. The sets $\mathcal{B}_0, \ldots, \mathcal{B}_4$ for the automaton $\mathcal{F}_5$ are shown in Fig. 11. We see that

$$\delta_{\mathcal{F}_h}(\mathcal{B}_i, 1) = \mathcal{B}_{i+1} \quad \text{for } i = 0, \ldots, h-2, \tag{24}$$
$$\delta_{\mathcal{F}_h}(\mathcal{B}_{h-1}, 1) = \mathcal{B}_{h-1}. \tag{25}$$

Also $\delta_{\mathcal{F}_h}(\mathcal{B}_i, 0) \subseteq \mathcal{B}_0$ for any $i < h$ (for $i < h-1$ it is even $\delta_{\mathcal{F}_h}(\mathcal{B}_i, 0) = \mathcal{B}_0$). Let us consider a configuration $\mathcal{C}$. If $\mathcal{C} \subseteq \mathcal{B}_i$ for some i then both $\delta_{\mathcal{F}_h}(\mathcal{C}, 0)$ and $\delta_{\mathcal{F}_h}(\mathcal{C}, 1)$ are subsets of some $\mathcal{B}_j$ and $\mathcal{B}_k$. By induction, for all strings s, $\delta_{\mathcal{F}_h}(\mathcal{L}'_h, s) \subseteq \mathcal{B}_i$ for some i (but i need not be unique).

Now, let us consider a merging string s_h for the set $\mathcal{L}'_h$ of $\mathcal{F}_h$. If s_h has a substring 01^i0 with $i < h-1$, this substring can be substituted by 00^i0 and the resulting string is still merging for $\mathcal{L}'_h$. Indeed, after a 0 the automaton is in a configuration $\mathcal{C}$ that is a subset of $\mathcal{B}_0$. Then the strings 1^i0 and 0^i0 both bring it to exactly the same configuration $\mathcal{C}' \subseteq \mathcal{B}_0$. If we denote $\alpha_0 = \rho$, the coin from α_k goes in both cases to $\alpha_{(k+i+1) \mod h}$ (see also Figs. 10 and 11).

As a result, if s is a merging string for the set $\mathcal{L}'_h$ of $\mathcal{F}_h$ then there is a merging string s' for $\mathcal{L}'_h$ of the same length with no substrings of the form 01^i0, $i < h-1$. We may also assume that the string s' does not start with 1^i0, $i < h-1$ either.

The following operations form the strings s' (the cost of each operation, i.e. the number of letters that form the operation, is also given):

1. From a subset of $\mathcal{B}_0$, the string 1^{h-1} brings the automaton to a configuration that is a subset of $\mathcal{B}_{h-1}$. The coins from ρ and α_1 are moved to ρ and from α_i, $i > 1$, to β_{i-1} (Fig. 12(a)). The cost of this operation is $h-1$.

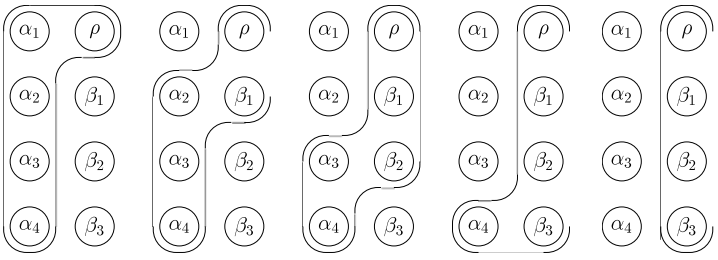


Fig. 11. The configurations $\mathscr{B}_i$ for the automaton $\mathscr{F}_5$.

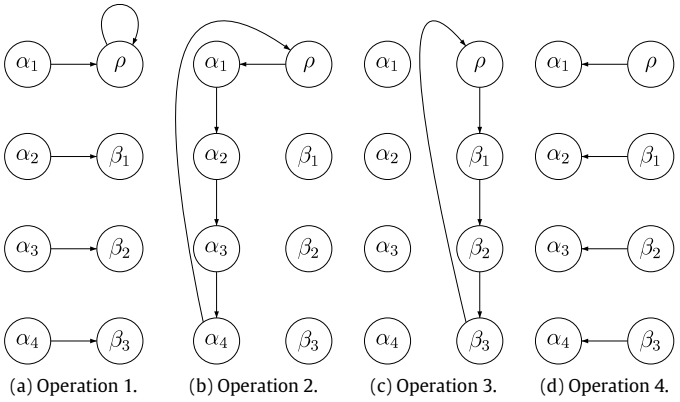


Fig. 12. Movements of coins under the operations 1–4.

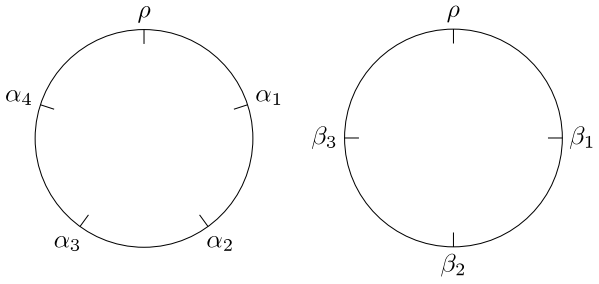


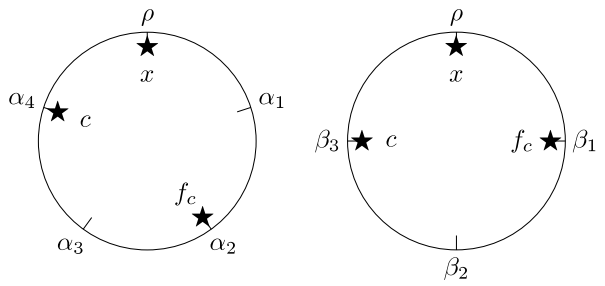
Fig. 13. An automaton equivalent to $\mathscr{F}_5$.

2. From a subset of $\mathscr{B}_0$, the letter 0 brings the automaton to another subset of $\mathscr{B}_0$. The coins from α_i move to α_{i+1} for $i = 1, \ldots, h-2$. The coin from ρ goes to α_1 and from α_{h-1} to ρ (Fig. 12(b)). The cost of this operation is 1.
3. From a subset of $\mathscr{B}_{h-1}$, the letter 1 brings the automaton to another subset of $\mathscr{B}_{h-1}$. The coins from β_i move to β_{i+1} for $i = 1, \ldots, h-3$. The coin from ρ moves to β_1 and from β_{h-2} to ρ (Fig. 12(c)). The cost of this operation is 1.
4. From a subset of $\mathscr{B}_{h-1}$, the letter 0 brings the automaton to a subset of $\mathscr{B}_0$. The coins from β_i, $i = 1, \ldots, h-2$, go to α_{i+1}. The coin from ρ moves to α_1 (Fig. 12(d)). The cost of this operation is 1.

With these operations we may forget about the sets $\mathscr{B}_1, \ldots, \mathscr{B}_{h-2}$ and consider only $\mathscr{B}_0$ and $\mathscr{B}_{h-1}$. The sets $\mathscr{B}_0$ and $\mathscr{B}_{h-1}$ can be visualized as circles with marked positions $\rho, \alpha_1, \ldots, \alpha_{h-1}$, for $\mathscr{B}_0$ (let us call it circle $\mathfrak{C}_1$), and $\rho, \beta_1, \ldots, \beta_{h-2}$, for $\mathscr{B}_{h-1}$ (let us call it circle $\mathfrak{C}_2$), as in Fig. 13. Operations 2 and 3 move all the coins in the clockwise direction on circle $\mathfrak{C}_1$ and $\mathfrak{C}_2$, respectively. Operation 1 transforms the circle $\mathfrak{C}_1$ (Fig. 13(a)), to the circle $\mathfrak{C}_2$ (Fig. 13(b)). This is done by merging ρ and α_1 in the new position ρ and then renaming α_i to β_{i-1} for $0 < i < h$. Operation 4 does the reverse. It first renames β_i to α_{i+1}, then renames ρ to α_1 and finally inserts a new empty position ρ between α_{h-1} and α_1.

The initial configuration, $\mathscr{L}'_h$, is the circle $\mathfrak{C}_1$ with coins on all positions. Now, the goal is to find a sequence of the operations 1–4 that merges the initial configuration, such that the total cost of the operations is minimal (the lightest merging sequence). It is easy to see that any such sequence moves all the coins to ρ.

Let the *distance* between two positions, η and θ, of the circle $\mathfrak{C}_i$ ($i = 1, 2$), $d(\eta, \theta)$, be the number of hops from η to θ in clockwise direction. The distance between two coins is the distance between their positions. The distance between a coin



(a) Coins on $\mathfrak{C}_1$. (b) Coins on $\mathfrak{C}_2$.

Fig. 14. Example of coins on circles $\mathfrak{C}_1$ and $\mathfrak{C}_2$ for $h = 5$.

c and a position η is the distance between the position of c and η, and similarly for the distance between a position and a coin. We assume that $d(\eta, \eta) = 0$.

Let the *value* of a coin c, d_c, be the largest distance to any other coin. The coin with the largest distance from c is denoted by f_c. We say that a coin (or a position) is *between* the coins (or positions) η and θ if it is on the path from η to θ in the clockwise direction, but it is neither on η nor on θ.

Example 22. In Fig. 14(a) there are three coins on the circle $\mathfrak{C}_1$. We have $d(c, \rho) = d(c, x) = 1$, $d(c, f_c) = 3$ and f_c is the farthest coin from c. The position ρ is between c and f_c, but is not between f_c and c (only α_3 is between f_c and c).

For each coin, c, we define a function, P_c, called the *potential* of c. It depends on the position of the coin c and positions of other coins. It also depends on whether the coins are in the circle $\mathfrak{C}_1$ or $\mathfrak{C}_2$.

The potential of a coin c in a configuration is defined for any configuration with at least two coins by:

$$P_c = P_{0c} + P'_c, \tag{26}$$

where:

$$P_{0c} = (d_c - 1)(2h - 1), \tag{27}$$

and

$$P'_c = \begin{cases} \text{case (i):} & d(c, \rho) & \text{if the coins are on } \mathfrak{C}_1, \\ \text{case (ii):} & d(c, \beta_{h-2}) + 2 & \text{if the coins are on } \mathfrak{C}_2 \text{ and } \rho \text{ is between } f_c \text{ and } c \text{ or } c \text{ is on } \rho, \\ \text{case (iii):} & d(c, \beta_{h-2}) + h + 1 & \text{if the coins are on } \mathfrak{C}_2 \text{ and } \rho \text{ is between } c \text{ and } f_c \text{ or } f_c \text{ is on } \rho. \end{cases} \tag{28}$$

The definition is valid if there are at least two coins. The value of P'_c is in $\{0, \ldots, h-1\}$ in case (i), in $\{2, \ldots, h\}$ in case (ii), and in $\{h+1, \ldots, 2h-1\}$ in case (iii).

Example 23. In Fig. 14(a) the potentials are the following.

$$\begin{aligned} P_c &= (d(c, f_c) - 1) \cdot (2h - 1) + d(c, \rho) = 2 \cdot 9 + 1 = 19 \\ P_x &= (d(x, c) - 1) \cdot (2h - 1) + d(x, \rho) = 3 \cdot 9 + 0 = 27 \\ P_{f_c} &= (d(f_c, x) - 1) \cdot (2h - 1) + d(f_c, \rho) = 2 \cdot 9 + 3 = 21. \end{aligned}$$

In Fig. 14(b) the potentials are the following.

$$\begin{aligned} P_c &= (d(c, f_c) - 1) \cdot (2h - 1) + d(c, \beta_3) + h + 1 \\ &= 1 \cdot 9 + 0 + 5 + 1 = 15 && \text{(case (iii))} \\ P_x &= (d(x, c) - 1) \cdot (2h - 1) + d(x, \beta_3) + 2 \\ &= 2 \cdot 9 + 3 + 2 = 23 && \text{(case (ii))} \\ P_{f_c} &= (d(f_c, x) - 1) \cdot (2h - 1) + d(f_c, \beta_3) + h + 1 \\ &= 2 \cdot 9 + 2 + 5 + 1 = 26 && \text{(case (iii)).} \end{aligned}$$

Let the *leader* be the coin with the lowest potential (we do not assume that the leader is unique, but we will later see that it is the case). The potential of a configuration is the potential of its leader.

In Fig. 14(a) the coin c is the leader. The potential of this configuration is equal to 19. In Fig. 14(b) also the coin c is the leader. The potential of this configuration is now equal to 15.

Lemma 24. *If c has minimal potential of all the coins in a given configuration then it also has minimal value d_c. Moreover, if two coins c_1 and c_2 have the same potential then they have the same value: $d_{c_1} = d_{c_2}$.*

Proof. If the configuration is on $\mathfrak{C}_1$ then $0 \le P'_c \le h - 1$. If the configuration is on $\mathfrak{C}_2$ then $2 \le P'_c \le 2h - 1$. In both cases the range of values of P'_c is less than $2h - 1$ and $2h - 1$ is the least difference in potential if d_c differs. This proves both statements of the Lemma. □

Lemma 25. *Each coin in a fixed configuration has a different potential.*

Proof. Let us assume the contrary, that there are two coins c_1 and c_2 with the same potential. From Lemma 24 we know that $d_{c_1} = d_{c_2}$, so $P'_{c_1} = P'_{c_2}$. If the configuration is on $\mathfrak{C}_1$ then P'_{c_i} is described by (i) and the coins have to be at the same position on the circle, which contradicts the assumption that they are different. The same happens if both P'_{c_1} and P'_{c_2} are defined by case (ii) or both of them are defined by case (iii). On the other hand, the ranges of P'_c for cases (ii) and (iii) of (28) are disjoint, so no other choice is possible. □

Lemma 26. *After each operation 1–4, assuming that there are at least two coins after the operations, the potential of a configuration decreases by at most the cost of the operation. Moreover, in each configuration there is an operation that decreases the potential of the configuration by exactly the cost of the operation, as long as after each operation there are at least two coins.*

Proof. Each possible operation will be considered separately. We will first show that the potential of any coin c decreases by at most the cost of the operation. Then, for any coin c we will find an operation O_c (not necessarily unique) that decreases the potential of c by exactly the cost of O_c. This will prove the lemma, because the potential of the leader c_L may decrease by the cost of O_{c_L}. The leader will also be the leader after the operation O_{c_L} because the potential of no other coin can decrease by a larger value after applying O_{c_L} and the leader had the lowest potential before the operation.

Let us consider the circle $\mathfrak{C}_1$ first:

- Operation 1: If d_c does not change then P'_c cannot decrease by more than $h-1$. Otherwise, d_c decreases by exactly one. We have to show that in this case P'_c increases by at least h. Before the operation we had $P'_c = d(c, \rho)$. As the distance between c and f_c decreased after the operation, there are two cases:
  - The first one is that c is on ρ before the operation. Then it stays on ρ after the operation as well. The new value of P'_c is described by case (ii) of (28), which means that it increased from 0 to $d(\rho, \beta_{h-2}) + 2 = h$.
  - The second case is that c is not on ρ before the operation. As the distance, d_c, decreased, f_c must have been between ρ and c. Thus, after the operation f_c is between ρ and c or f_c is on ρ. The new value of the potential is described by case (iii) of (28): $P'_c = d(c, \beta_{h-2}) + h + 1$. Since c is not on ρ, $d(c, \beta_{h-2}) = d(c, \rho) - 1$ and P'_c increases by exactly h.
- Operation 2: The potential of a coin c either decreases by one, if c is not on ρ, or increases by $h-1$, if c is on ρ. Thus, in both cases the potential decreases by at most one.

If the coin c is not on ρ, operation 2 decreases the potential of c by 1. Otherwise, i.e. if c is on ρ, operation 1 decreases the potential of c by $h-1$.

For the circle $\mathfrak{C}_2$:

- Operation 3: If P'_c before and after the operation are both described by the same case (ii) or (iii) of (28) then the potential decreases by at most one. If before the operation the potential is described by case (ii) and afterwards by case (iii), it can only increase (note that $h \geq 3$). Finally, if the potential before the operation is described by case (iii) and afterwards by case (ii) then at start c is on β_{h-2} and afterwards c is on ρ. Then the difference in the potential is:

$$\begin{aligned} \{P'_c\}_{\text{after}} - \{P'_c\}_{\text{before}} &= d(\rho, \beta_{h-2}) + 2 - d(\beta_{h-2}, \beta_{h-2}) - h - 1 \\ &= h - 2 + 2 - 0 - h - 1 = -1, \end{aligned} \tag{29}$$

  which is correct.
- Operation 4: If d_c increases after applying the operation then P_c increases as well. Otherwise initially c is on ρ or ρ is between f_c and c. In both circumstances P'_c is described by case (ii) of (28). At start $P'_c = d(c, \beta_{h-2}) + 2$. After the operation: $P'_c = d(c, \rho)$ (case (i) of (28)). As c cannot be on ρ after the operation, $\{d(c, \beta_{h-2})\}_{\text{before}} = \{d(c, \rho)\}_{\text{after}} - 1$, and

$$\{P'_c\}_{\text{after}} - \{P'_c\}_{\text{before}} = \{d(c, \rho)\}_{\text{after}} - \{d(c, \beta_{h-2})\}_{\text{before}} - 2 = -1, \tag{30}$$

  thus the potential decreased by exactly one.

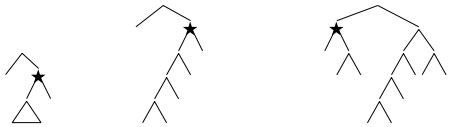
Operation 4 decreases the potential of c by one if the potential is described by the case (ii) of the equation. Otherwise, operation 3 decreases the potential of c by 1.

Note that in many cases the operation mentioned in the lemma is not unique. □

Proof of Theorem 21. The length of the shortest merging sequence s for the set $\mathscr{L}'_h$ of states of $\mathscr{F}_h$ is the cost of the lightest merging sequence s_o of operations 1–4 for $\mathscr{L}'_h$. The sequence s can be obtained from s_o by substituting the operations by their corresponding string. We have already stated that such a sequence s also merges the set $\mathscr{L}_h$ of states of D_h.

Operation 1 is the only one that reduces the number of coins. So the last operation of s_o is operation 1, that merges the coins in ρ and α_1. The cost of this final operation is $h-1$. Let the configuration with coins just in ρ and α_1 be denoted by $\mathscr{F}$ (for almost-Final).

By Lemma 26 it is always possible to decrease the potential of a configuration by the cost of an operation as long as the final configuration has at least two coins. The only configuration that may lead from a configuration with at least two coins to a configuration with just one coin is the configuration $\mathscr{F}$, and $\mathscr{F}$ has the lowest potential of all configurations with at least



(a) A tree G_k. (b) G_4, $s = 0^4$. (c) C_4, $s = 1110^4$ or $s = 1010^4$.

Fig. 15. The nodes with the longest merging string for the two families C_k and G_k.

two coins. It is then always possible to reach the configuration $\mathscr{F}$ starting from any configuration with at least two coins and using only optimal operations, i.e. the operations that have the cost equal to the drop in the potential.

The potential of the initial configuration is

$$P_0 = 0 + (h-2)(2h-1) = 2h^2 - 5h + 2. \tag{31}$$

The potential of the configuration $\mathscr{F}$ is $P_1 = 0$. There is a sequence s_o of operations 1–4 of cost $P_0 - P_1$ that leads from the initial configuration to the configuration $\mathscr{F}$. No sequence leading from the initial configuration to the configuration $\mathscr{F}$ can have lower cost. Thus, the length of the shortest merging string s for the set $\mathscr{L}'_h$ of $\mathcal{F}_h$ is

$$|s| = P_0 - P_1 + h - 1 = 2h^2 - 4h + 1. \tag{32}$$

The sequence s is also a synchronizing sequence for the automaton $\mathcal{F}_h$. Indeed, any shortest merging string for the set $\mathscr{L}'_h$ has to start with 1^{h-1}. The application of 1^{h-1} to either the set of all states of $\mathcal{F}_h$ or to the set $\mathscr{L}'_h$ results in the same configuration: the coins on the states $\rho, \beta_1, \ldots, \beta_{h-2}$.

Finally, by Theorem 11, the synchronizing string for $\mathcal{F}_h$ we found is also a synchronizing string for D_h. D_h cannot have any shorter synchronizing string, because it would also be a merging string for the set $\mathscr{L}'_h$ of states of $\mathcal{F}_h$, which is impossible.

The minimized Huffman automaton for D_h has $K = 2(h-1)$ nodes. Even though it contains a letter that reduces the number of coins by $h - 2 = \frac{K}{2} - 1$ (a letter of deficiency $\frac{K}{2} - 1$), its shortest synchronizing string is of length $2h^2 - 4h + 1 = \frac{K^2}{2} - 1$, which is quadratic in K. This makes the automata D_h interesting in themselves.

The results of the search allow us to state the following conjecture.

Conjecture 27. *The length of the shortest synchronizing string s for a code with N codewords, $N \geq 9$, with h being the length of the longest codeword, is at most:*

$$|s| \leq \min(2N - a, 2h^2 - 4h + 1), \tag{33}$$

where a is 7 for odd N and 9 for even N.

5.2. Long merging string

For trees of fixed number of leaves, N, the length of the shortest merging string for $\{n, \varepsilon\}$ in the worst case is equal to $N - 2$, for $N = 3, \ldots, 20$, apart from $N = 6$. For $N = 6$ the worst-case length is equal to $N - 1 = 5$. Two families of trees have the worst-case shortest merging strings. The first one corresponds to the code

$$G_k = \{0, 10^k\} \cup \{10^i 1 | i < k\}, \quad k \geq 1, \tag{34}$$

and gives the worst-case trees for N from 3 to 20, apart from $N = 6$. The number of leaves of the tree G_k is $N = k + 2$. The merging string for the set $\{1, \varepsilon\}$ is of length $N - 2$. The structure of these trees is shown in Fig. 15(a). The tree G_4 is shown in Fig. 15(b). These figures also indicate the node whose merging string with ε is the longest.

The other family of trees is the family C_k (see (20) and Fig. 6(a)) with even k (odd number of codewords). The merging string for the set $\{0, \varepsilon\}$ is of length $N - 2$ and this is the worst case for $N = 7, 9, 11, \ldots 19$. The node with the longest merging string with ε is 0 (Fig. 15(c)).

There are also additional worst-case trees for $N = 5, 6, 7, 9, 10, 12$. They correspond to neither the trees C_k nor G_k. They are presented in Fig. 16. The nodes with the longest merging string with the root are marked with a star.

The worst-case trees among trees of fixed height, h, are the trees D_h (Eq. (21) and Fig. 9).

Theorem 28. *The upper bound on the length of the shortest merging string for any pair $\{n, \varepsilon\}$, where n is a state of D_h, is $\lceil h^2 - \frac{3}{2}h \rceil$. For odd h it is achieved by the pair $\{0^{(h-1)/2}, \varepsilon\}$. For even h it is achieved by pairs $\{x, \varepsilon\}$, where x is any binary string of length $\frac{h}{2}$ containing at least one 0.*

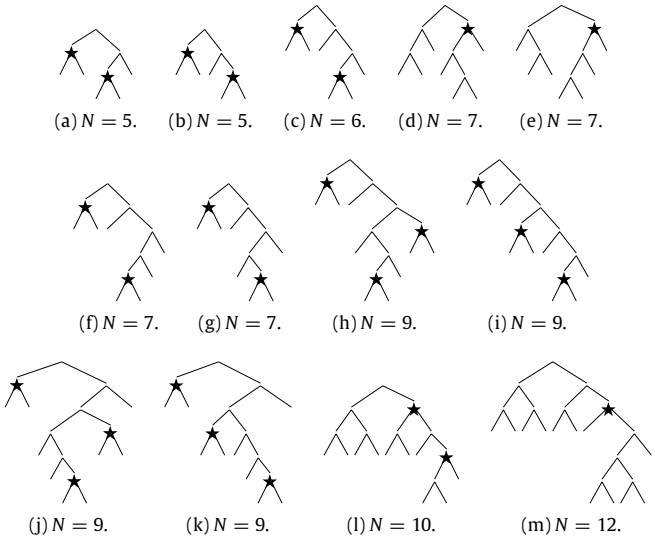


Fig. 16. Worst-case trees in terms of the length of the shortest merging string. The nodes that require a merging string of the worst-case length are marked with a star. The trees of the families C_k and G_k are omitted.

Proof. The length of the shortest merging string is given by the potential of the position plus $h - 1$ (see the proof of Theorem 21). It is enough to find a two-coin configuration with one coin on ρ that has the largest potential.

For even h, the maximum of the minimal distance between the two coins is $h/2$ and it is possible only in the circle $\mathfrak{C}_1$. The second coin is then on $\alpha_{h/2}$. In this configuration the coin with minimal potential is the one on ρ. The length of the shortest merging string is:

$$\begin{aligned} |s| &= P_\rho + (h-1) = P_{0\rho} + P'_\rho + (h-1) \\ &= (d_\rho - 1)(2h-1) + d(\rho, \rho) + (h-1) \\ &= (\tfrac{h}{2} - 1)(2h-1) + 0 + (h-1) \\ &= h^2 - \tfrac{3}{2}h. \end{aligned}$$

For odd h, the maximum of the minimal distance between coins is $(h-1)/2$. Such a configuration is possible on either the circle $\mathfrak{C}_1$ and $\mathfrak{C}_2$. Let us assume for a moment that the coins are in $\mathfrak{C}_2$. The second coin must be $\beta_{(h-1)/2}$ to get the distance $(h-1)/2$ between the coins. The coin with minimal potential is the one on ρ. The length of the shortest merging string is:

$$\begin{aligned} |s| &= P_\rho + (h-1) = P_{0\rho} + P'_\rho + (h-1) \\ &= (d_\rho - 1)(2h-1) + d(\rho, \beta_{h-2}) + 2 \\ &= \left(\tfrac{h-1}{2} - 1\right)(2h-1) + (h-2) + 2 + (h-1) \\ &= h^2 - \tfrac{3}{2}h + \tfrac{1}{2}. \end{aligned}$$

No potential of any position on $\mathfrak{C}_1$ can be higher, because the value of P'_c on the circle $\mathfrak{C}_1$ never exceeds $h - 1$. (It is equal to h in the position on $\mathfrak{C}_2$ analyzed above.) □

The results of the search allow us to state the following conjecture.

Conjecture 29. *For any Huffman automaton $\mathcal{T}$, corresponding to a code with N codewords, with h being the length of the longest codeword, the length of the shortest merging string, s_n, for a set $\{n, \varepsilon\}$, where n is any state of $\mathcal{T}$ is bounded by:*

$$|s_n| \leq \min(N - 2, \lceil h^2 - \tfrac{3}{2}h \rceil), \tag{35}$$

if $N \neq 6$, and $|s| \leq 5$ for $N = 6$.

6. Summary

We presented a constructive upper bound on the length of the shortest merging string and the shortest synchronizing string for a Huffman code.

We tested the lengths of the shortest merging and synchronizing string on all codes of size from 3 to 20 and on all codes with the length of the longest codeword from 2 to 5. Three classes of worst-case codes were found. The length of the shortest synchronizing strings for these classes of codes is far from the bound proven before. This allowed us to formulate conjectures, which remain open.

Acknowledgements

The authors' research was partially supported by the grants of the Polish Ministry of Science and Higher Education N N206 376134 (M.T. Biskup) and N 206 004 32/0806 (W. Plandowski).

References

[1] Dimitry S. Ananichev, Mikhail V. Volkov, Yu.I. Zaks, Synchronizing automata with a letter of deficiency 2, Theor. Comput. Sci. 376 (1–2) (2007) 30–41.
[2] Dimitry S. Ananichev, Mikhail V. Volkov, Synchronizing generalized monotonic automata, Theor. Comput. Sci. 330 (1) (2005) 3–13.
[3] Renato M. Capocelli, Luisa Gargano, Ugo Vaccaro, On the characterization of statistically synchronizable variable-length codes, IEEE Trans. Inform. Theory 34 (4) (1988) 817–825.
[4] Renato M. Capocelli, Alfredo De Santis, Luisa Gargano, Ugo Vaccaro, On the construction of statistically synchronizable codes, IEEE Trans. Inform. Theory 38 (2) (1992) 407–414.
[5] Ján Černý, Poznámka k. homogénnym experimentom s konecnými automatmi, Mat. Fyz. čas SAV 14 (1964) 208–215.
[6] David Eppstein, Reset sequences for monotonic automata, SIAM J. Comput. 19 (3) (1990) 500–510.
[7] Thomas J. Ferguson, J.H. Rabinowitz, Self-synchronizing Huffman codes, IEEE Trans. Inform. Theory 30 (4) (1984) 687–693.
[8] Christopher F. Freiling, Douglas S. Jungreis, François Théberge, Kenneth Zeger, Almost all complete binary prefix codes have a self-synchronizing string, IEEE Trans. Inform. Theory 49 (9) (2003) 2219–2225.
[9] Yuh-Ming Huang, Sheng-Chi Wu, Shortest synchronizing codewords of a binary Huffman equivalent code, in: ITCC'03: Proceedings of the International Conference on Information Technology: Computers and Communications, IEEE Computer Society, Washington, DC, USA, 2003, p. 226.
[10] Jarkko Kari, Synchronizing finite automata on eulerian digraphs, Theor. Comput. Sci. 295 (1–3) (2003) 223–232.
[11] J.C. Maxted, John P. Robinson, Error recovery for variable length codes, IEEE Trans. Inform. Theory 31 (6) (1985) 794–801.
[12] Stephanie Perkins, Adrian Escott, Synchronizing codewords of q-ary Huffman codes, Discrete Math. 197–198 (1999) 637–655.
[13] Jean-Eric Pin, On two combinatorial problems arising from automata theory, Ann. Discrete Math. 17 (1983) 535–548.
[14] Beulah Rudner, Construction of minimum-redundancy codes with an optimum synchronizing property, IEEE Trans. Inform. Theory 17 (4) (1971) 478–487.
[15] Sven Sandberg, Homing and synchronizing sequences, in: Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, Alexander Pretschner (Eds.), Model-Based Testing of Reactive Systems, in: Lecture Notes in Computer Science, vol. 3472, Springer, 2004, pp. 5–33.
[16] Marcel Paul Schützenberger, On synchronizing prefix codes, Inf. Control 11 (4) (1967) 396–401.
[17] Avraham Trahtman, An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture, in: Rastislav Kralovic, Pawel Urzyczyn (Eds.), MFCS, in: Lecture Notes in Computer Science, vol. 4162, Springer, 2006, pp. 789–800.